REF AL

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開平10－91448
(43)公開日　平成10年(1998)４月10日

(51)Int.Cl.⁶	識別記号	ＦＩ	
Ｇ０６Ｆ　9/44	５３０	Ｇ０６Ｆ　9/44	５３０Ｐ ５３０Ｍ

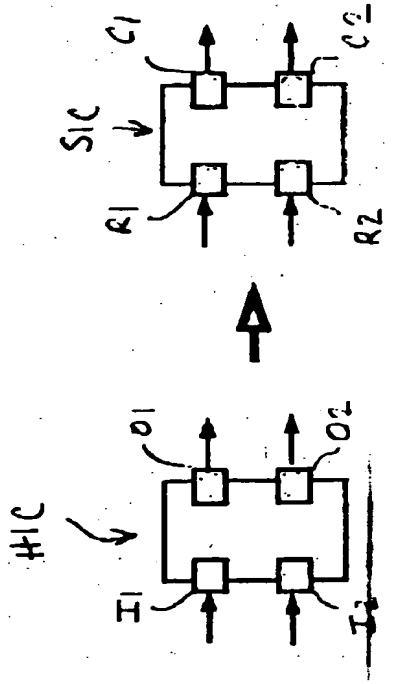
審査請求　未請求　請求項の数18　ＯＬ　（全９頁）

(21)出願番号　特願平9－178320
(22)出願日　平成９年(1997)７月３日
(31)優先権主張番号　０８／６７５８４６
(32)優先日　1996年７月３日
(33)優先権主張国　米国（ＵＳ）

(71)出願人　390039413
シーメンス　アクチエンゲゼルシヤフト
ＳＩＥＭＥＮＳ　ＡＫＴＩＥＮＧＥＳＥＬＬＳＣＨＡＦＴ
ドイツ連邦共和国　ベルリン　及び　ミュンヘン　（番地なし）
(72)発明者　ハンス－エーリッヒ　ラインフェルダー
ドイツ連邦共和国　エアランゲン　プラーテンシュトラーセ　23
(72)発明者　カールハインツ　ドルン
ドイツ連邦共和国　カルヒロイト　エアレンシュトラーセ　29
(74)代理人　弁理士　矢野　敏雄　（外１名）
最終頁に続く

(54)【発明の名称】　オブジェクト指向コンピューティングシステム

(57)【要約】
【課題】　大きなアプリケーションへの、意味構造のないソフトウェアコンポーネントまたはビルディングブロックを、ビルディングブロック内でのコード変化なしで、そしてアダプタを書くことなしに独立して結合するための方法および／または手段を提供する。
【解決手段】　意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有するようにする。



【特許請求の範囲】
【請求項１】　意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、
自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とするオブジェクト指向コンピューティングシステム。
【請求項２】　オブジェクトの入力と出力はそれぞれ、CsaConnectableとCsaRemoteオブジェクトを介して供給される、請求項１記載のオブジェクト指向コンピューティングシステム。
【請求項３】　入力と出力に関連した各データ構造が別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項２記載のオブジェクト指向コンピューティングシステム。
【請求項４】　各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力と出力のASCII構築ファイル名のランタイムでダイナミックにリンク可能である、請求項２記載のオブジェクト指向コンピューティングシステム。
【請求項５】　ダイナミックにリンク可能な入力および出力と、オブジェクトへおよびオブジェクトからそれぞれ前記入力および出力を介して伝送されるデータをキューイングするための内部タスクと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とするオブジェクト指向コンピューティングシステム。
【請求項６】　オブジェクトの入力と出力は、CsaConnectableオブジェクトとCsaRemoteオブジェクトをそれぞれ介して供給される、請求項５記載のオブジェクト指向コンピューティングシステム。
【請求項７】　入力および出力に関連する各データ構造が別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項６記載のオブジェクト指向コンピューティングシステム。
【請求項８】　各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力と出力の名前を含むASCII構築ファイルによるランタイムでダイナミックにリンク可能である、請求項６記載のオブジェクト指向コンピューティングシステム。
【請求項９】　オブジェクト指向コンピューティングシステムに含まれるソフトウェアコンポーネントを設計するための方法において、
十分に分散可能な入力と出力を定義し、
リンク可能であり、意味構造のないソフトウェアコンポーネントを入力および出力接続点によって構築し、
十分に分散可能なイベントに基づくオートルーティングされるパターンを、イベント通信フレームワークに基づいて供給する、ことを特徴とする方法。
【請求項１０】　コンピュータプラットホーム上にオブジェクト指向コンピューティングシステムを有するオブジェクト指向コードを含む記憶媒体において、
意味構造のない、ダイナミックにリンク可能な入力および出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とする記憶媒体。
【請求項１１】　オブジェクトの入力と出力は、CsaConnectableオブジェクトとCsaRemoteオブジェクトを介してそれぞれ供給される、請求項１０記載の記憶媒体。
【請求項１２】　入力および出力に関連する各データ構造は別個のヘッダファイルに記述されており、該記述ファイルはリンクすべき各オブジェクトにより使用される、請求項１１記載の記憶媒体。
【請求項１３】　各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力および出力の名前でファイルされたASCIIコンフィギュレーションによってランタイムでダイナミックにリンク可能である、請求項１１記載の記憶媒体。
【請求項１４】　コンピューティングシステム上のオブジェクト指向コンピューティングシステムに対するオブジェクト指向コードを含む記憶媒体において、
ダイナミックにリンク可能な入力および出力と、
オブジェクトへおよびオブジェクトから前記入力および出力をそれぞれ介して伝送されるデータのキューイングのため内部タスクと、
自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とする記憶媒体。
【請求項１５】　オブジェクトの入力および出力は、CsaConnnectableオブジェクトおよびCsaRemoteオブジェクトを介してそれぞれ供給される、請求項１４記載の記憶媒体。
【請求項１６】　入力および出力に関連する各データ構造は別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項１５記載の記憶媒体。
【請求項１７】　各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力および出力の名前を含むASCII構築ファイルにより、ランタイムでダイナミックにリンク可能である、請求項１５記載の記憶媒体。
【請求項１８】　オブジェクト指向コンピューティングシステムでソフトウェアコンポーネントを設計するための方法に対するオブジェクト指向コードを含む記憶媒体において、
十分に分散可能な入力および出力イベントを定義し、
ダイナミックにリンク可能で、意味構造のないソフトウ

ェアコンポーネントを入力および出力接続点により構築し、
十分に分散可能なイベントに基づいてオートルーティングされるパターンをイベント通信フレームワークに基づいて供給する、ことを特徴とする記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、オブジェクト指向マルチプログラミングシステムに関する。より具体的には、本発明は、ソフトウェアコンポーネントまたはビルディングブロックを相互接続するための方法および手段に関するものである。
【０００２】
【従来の技術】米国特許第５４９９３６５号明細書に記載されているように、オブジェクト指向プログラミングシステムおよびプロセス（“オブジェクト指向コンピュータ環境”とも称される）は、多くの研究と興味の対象である。当業者には周知であるように、オブジェクト指向プログラミングシステムは多数の“オブジェクト”から構成される。１つのオブジェクトはデータ構造であり、“フレーム”と称される。演算または関数のセットはまた“メソッド”とも称され、データ構造にアクセスすることができる。フレームは“スロット”を有することができ、各スロットは、スロット内のデータの“属性”を含む。この属性は（整数またはストリングのような）基関数、または他のオブジェクトに対するポインタであるオブジェクトレファレンスとすることができる。識別データ構造と共通の特性を有するオブジェクトは相互に群に入れることができ、１つの“クラス”として正しく識別される。
【０００３】オブジェクトの各所定のクラスは通常、“インスタンス”の数に現れる。各インスタンスは所定のデータ構造をオブジェクトの所定の例に対して含む。オブジェクト指向コンピュータ環境ではデータは次のように処理される。すなわち、オブジェクトにそのメソッドの１つを、オブジェクト“メッセージ”を送信することによって実行することを要求することにより処理される。受信オブジェクトはメッセージに次のようにして応答する。すなわち、メッセージネームを満たすメソッドを選択し、このメソッドを名付けられたメソッドに基づいて実行し、コントロールをコーリング・ハイレベル・ルーティンに、このメソッドの結果に沿って戻すのである。クラス間の関係、オブジェクトおよびインスタンスは伝統的に“形成時間”中、またはオブジェクト指向コンピュータ環境の発生中に確立される。すなわち、“ランタイム”より時間的に前、またはオブジェクト指向コンピュータ環境の実行前に確立される。
【０００４】上記のクラス、オブジェクトおよびリファレンス間の関係に加えて、継承関係もまた２つまたはそれ以上のクラス間に存在する。すなわち、第１のクラスは第２のクラスの“ペアレント”と見なすことができ、第２のクラスは第１のクラスの“チャイルド”と見なすことができる。言い替えると、第１のクラスは第２のクラスの祖先であり、第２のクラスは第１のクラスの子孫である。従って第２のクラス（すなわち子孫）は第１のクラス（すなわち祖先）から引き継がれる。チャイルドクラスのデータ構造はペアレントクラスの属性のすべてを含む。
【０００５】オブジェクト指向システムはこれまで、オブジェクトの“バージョン”であると認識されていた。オブジェクトのバージョンは、異なる時点でのオブジェクトで同じデータである。例えば、“ワーク・イン・プログレス”に関連するオブジェクトは完成した立証済みのワークに関連する同じオブジェクトとは別個のバージョンである。多くのアプリケーションはまた、データがそれぞれ異なる時点で存在したというデータの歴史的記録を要求する。従ってオブジェクトの別のバージョンが要求される。
【０００６】２つの刊行物がさらに一般的背景を示している。E.W.Dijkstra, The Structure of "THE" Multi programming System, Communications of the ACM, Vol. 11,No.5, May 1968, pp.341-346,とC.A.R.Hoare, Monitors: Operating Systems Structuring Concepts, Communications of the ACM, Vol.17, No.10, October 1974, pp.549-557である。両方の刊行物は互いを引用している。前者の刊行物は、プリミティブを使用した同期方法について記載し、セマフォーの使用を説明するものであり、後者の刊行物はモニタのBrinch-Hansensコンセプトをオペレーティングシステムの構築方法として開発したものである。とりわけ、Hoareの刊行物は、プロセスに対する同期の形態を紹介し、セマフォーに関し可能な移植法について述べており、プルーフルールについても実例を示している。
【０００７】Hoareの刊行物に記載されているように、オペレーティングシステムの第１の目的は、計算機設置をそのリソースに対する要求が予測不能である多くのプログラム間でシェアすることである。従って設計者の第１の役目は、種々の種類のリソース（例えば主記憶装置、ドラム記憶装置、磁気テープ・ハンドラ、コンソール）に対するスケジュールアルゴリズムによるリソースアロケーションをデザインすることである。この役目を簡単にするため、プログラマはリソースの各クラスごとに別個のスケジューラを構成しようと試みる。この場合各スケジューラは所定量のローカル管理データからなり、いくつかのスケジューラおよびファンクションと共に構成される。これらスケジューラおよびファンクションは、プログラムによりリソースを獲得および解放するために呼び出される。このような関連したデータとプロシージャの収集はモニタとして知られる。
【０００８】適応通信環境（ＡＣＥ）は、Douglas C. S

chmidt (Assistant Professor withthe Department of Computer Science, School of Engineering and Applied Science, Washington University) により開発されたネットワークプログラミングシステムのオブジェクト指向タイプである。ACEは、ユーザーレベルユニットとWIN32 (WindowsNTおよびWindows95) OSメカニズムを、形式安全で、効率的で、オブジェクト指向インターフェースによって1つにまとめる。

【0009】・ IPCメカニズムーインターネットドメインとUNIXドメイン・ソケット、TLI、名前付きパイプ (UNIXとWIN32に対し) およびSTREAMパイプ；

・ イベントマルチプレクスーUNIXでselect()とpoll()により、WIN32でWaitForMultipleObjectsにより；

・ ソラリススレッド、POSIX Pスレッド、およびWIN32スレッド；

・ 明示的ダイナミック・リンキング機構-例えばUNIXのdlopen/dlsym/dlclose、WIN32のLoadLibrary/GetProc；

・ メモリマップファイル；

・ システムVIPCーメモリのシェア、セマフォー、メッセージキュー；

・ サンRPC (GNU C++)

付加的に、ACEは多数のハイレベルクラスカテゴリーとネットワークやうろくらみんぐフレームワークを、ローレベルC++ラッパを統合し、強調するために有している。ACEのハイレベルコンポーネントは、コンカレント・ネットワーク・デーモンをサポートする。これらのデーモンはアプリケーションサービスからなる。ACEは最近、多数の商用プロダクツで使用される。この商用プロダクツは、ATMシグナリングソフトウェアプロダクツ、IBXモニタアプリケーション、ネットワークマネージメント、および移動通信システムと事業拡大ー分散医用システムに対する一般的ゲートウェイ通信を含む。ACEについての詳細な情報と文献は、WWWトで次のURLから得ることができる。

【0010】http://www.cs.wustl.edu/...schmidt/ACE-overview.html

以下の省略形が本出願で使用される。

【0011】スレッド　ープロセス内での平行実行ユニット。モニタ同期、強制的逐次化により、複数同時実行スレッドの並列アクセス。このスレッドはすべてモニタによりプロテクトされている1つのオブジェクトの機能をコールアップする。

【0012】・ 同期化プリミティブ　ーパラレルアクティビティの反復的正当化に対するオペレーティングシステムの手段

・ セマフォー　ーパラレルアクティビティに対する同期化プリミティブ

・ Mutex　ーパラレルアクティビティに対する専用同期化プリミティブ、
相互実行目的のために、これはクリティカルなコードレンジを含む。

【0013】・ コンディションキュー　ー所定の条件を参照するパラレルアクティビティに対するキューを待つイベント

・ ゲートロック　ー各エントリーファンクションに対するモニタのMutex、オブジェクトをプロテクションするため、オブジェクトの1つのエントリールーチンを使用するため同時には1つのパラレルアクティブティだけを許容するため。

【0014】・ ロングタームスケジュール　ーコンディションキュー内またはパラレルアクティビティに対するキュー待ちイベント内での1つのパラレルアクティビティの長期遅延。

【0015】・ ブローカ　ーディストリビュータ
付加的に次の省略形が使用される。

【0016】AFM　Asynchronous Function Manager (非同期ファンクションマネージャ)

SESAM　Service &; Event Synchronous Asynchronous Manager (サービスおよびイベント同期非同期マネージャ)

PAL　Programmable Area Logic (プログラマブルエリアロジック)

API　Application Programmers Interface (アプリケーション・プログラマー・インターフェース)

IDL　Interface Definiton Language (インターフェース定義言語)

ATOMIC　Asynchron Transport Optimizing observer-pattern-like systemsupporting several Modes (client/server-push/pull) for an IDL-less Communication subsystem (IDLレス通信システムに対する非同期伝送最適オブザーバパターン様システムサポート複数モード)

XDR　External Data Representation

I/O　入力/出力

CSA　Common Software Architecture (a Siemens AG computing system convention) (共通ソフトウェアアーキテクチュア) (シーメンス社コンピューティングシステム規定)

SW　ソフトウェア (Software)

過去において、ソフトウェアコンポーネントまたはビルディングブロックのインターフェースはアプリケーションプログラムインターフェース (API) にコード化するのが困難であった。この解決手段はプロセスにリンク可能であったが、しかしロケーション透過型ではなかった。さらにインターフェースは、コード化の困難なオブジェクトリファレンスを備えたインターフェース定義言語 (IDL) によって行われていた。

【００１７】
【発明が解決しようとする課題】本発明の課題は、大きなアプリケーションへの、意味構造のないソフトウェアコンポーネントまたはビルディングブロックを、ビルディングブロック内でのコード変化なしで、そしてアダプタを書くことなしに独立して結合するための方法および／または手段を提供することである。
【００１８】
【課題を解決するための手段】上記課題は本発明により、意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有するように構成して解決される。
【００１９】
【発明の実施の形態】そのために、コンポーネントまたはビルディングブロックの機能は周りの環境から完全に分離され、これにより同じプロセス内でロケートする必要がなく、その代わりにネットワークを介して分散することができ、その際にインターフェース定義言語を必要としない。
【００２０】実施例では、ソフトウェアコンポーネントを設計するための本発明の方法は次のステップを有する：すなわち、十分に分散可能な入力イベントおよび出力イベントを定義し；ダイナミックにリンク可能であり、意味構造のないソフトウェアコンポーネントを入力および出力接続点により構築し；十分に分散可能なイベントに基づくオートルーティングパターンを、イベント通信フレームワークに基づいて設定する。
【００２１】別の実施例では、本発明はオブジェクト指向コンピューティングシステムを提供する。このシステムは、意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、オートルーティングされ、パターンに基づき、十分に分散可能なイベントを設定するイベント通信フレームワークを有する。
【００２２】別の実施例では、オブジェクトの入力と出力はCsaConnectableオブジェクトとCsaRemoteオブジェクトへのリンクによってそれぞれ設定される。
【００２３】別の実施例では、入力および出力と関連した各データ構造が別個のヘッダファイルに記述され、このヘッダファイルをリンクすべきすべてのオブジェクトが使用することができる。
【００２４】別の実施例では、各オブジェクトは共有ライブラリであり、オブジェクトの入力および出力のASCII構築ファイル名によってランタイムでダイナミックにリンクすることができる。
【００２５】
【実施例】本発明を以下、有利な実施例に基づいて詳細に説明する。
【００２６】前に述べたように本発明は、意味構造のないビルディングブロックを大きなアプリケーションに、ビルディングブロック内でのコード変化なしで、かつアダプタを書くことなしで独立して結合するための手段である。このことにより、ソフトウェアブロックを、集積回路を結合できるのと同じ仕方で結合することのできるシステムまたはアーキテクチュアが得られる。
【００２７】本発明の目的のために、ソフトウェアビルディングブロックはその外環境（これはユーザーライトコードと同じように、１つ以上の他のビルディングブロックからなることができる）と、CsaConnectable（提供者）およびCsaRemote（消費者）オブジェクト／クラスを介して接続される。これは、他の接続終点が同じプロセスにあるかまたはリモートホスト（ローカルの場合は最適化により）にあるかは関係ない。このルールはすべてのビルディングブロックに適用される。
【００２８】ビルディングブロックの入力および出力と関連する各データ構造は別個のヘッダファイルに記述される。このヘッダファイルは接続すべきすべてのビルディングブロックが使用できる。
【００２９】各ビルディングブロックは共有ライブラリとして実現または構成される。この共有ライブラリは、パブリックドメイン通信パッケージで、ASCII構築ファイルが使用されるのと同じようにランタイムでダイナミックにリンクされる。入力および出力の名前はダイナミックリンキングの間に割り当てられ、これにより再コンパイルまたは再リンキングなしでコンフィギュレーションを変化することができる。
【００３０】この手段の大きな利点は、フレキシブルであることと、ハイレベルパターンが他のシンプルな、よくデザインされた、意味構造のないパターンの最適な結合から得られることである。さらにこの機構は、ハードウェアの世界のオンボード集積回路の結合と非常によく似ている。
【００３１】図１は、ハードウェア集積回路（ＩＣ）と本発明のソフトウェアオブジェクトとの類似性を比較するのに有用である。図１では、ハードウェアＩＣ　ＨＩＣが２つの入力ピン１１，１２と、２つの出力ピンＯ１，Ｏ２を有する。同じようにソフトウェアオブジェクトＳＩＣは、CsaRemoteを介する２つの入力Ｒ１，Ｒ２と、CsaConnectableを介する２つの出力Ｃ１，Ｃ２を有する。
【００３２】このようなソフトウェアＩＣシステムの実施のための符号化の例を以下に示す。
【００３３】１．入力／出力クラス定義
【００３４】
【外１】

```
#ifndef SAMPLECLASS1H
#define SAMPLECLASS1H
/*****************************\
*                             *
*     Input/Output data structure   *
*                             *
\*****************************/
struct SampleClass1 {
  int               theInteger;
  DECLARE_MSC (SampleClass1)
};
IMPLEMENT_MSC  (SampleClass1, V(theInteger))
#endif // SAMPLECLASS1H
#ifndef SAMPLECLASS2H
#define SAMPLECLASS2H
/*****************************\
*                             *
*     Input/Output data structure   *
*                             *
\*****************************/
struct SampleClass2 {
  int               theInteger;
  DECLARE_MSC (SampleClass2)
};
IMPLEMENT_MSC (SampleClass2, V(theInteger))
#endif // SAMPLECLASS2H
```

【００３５】ＩＩ．ビルディングブロックヘッダファイル

【００３６】
【外２】

```
#include<ace/Service_Object.h>
#include<CsaConnectable.hh>
#include<CsaRemote.hh>
#include<SampleClass1.h>
#include<SampleClass2.h>
class SampleApplication : public ACE_Service_Object
{
  public:
      virtual int init (int, char**);
      virtual int fini (void);
      virtual int info (char**, size_t) const;
      SampleApplication (void);
      ~SampleApplication (void);
  protected:
      CsaConnectable <SampleClass1> *output1;
      CsaConnectable <SampleClass2> *output2;
      CsaRemote <SampleClass1> *input1;
      CsaRemote <SampleClass2> *input2;
};
#endif // SAMPLE_APPLICATION
```

【００３７】ＩＩＩ．ビルディングブロック実施
【００３８】
【外３】

```
#include<CsaConnectable.hh>
#include<CsaRemote.hh>
#include<SampleApplication.h>
int SampleApplication : : init(int argc, char **argv)  {
    cout << endl << "Initializing " << endl;
    input1 = new CsaRemote <SampleClass1> (argv[1]);
    input2 = new CsaRemote <SampleClass2> (argv[2]);
    output1 = new CsaConnectable <SampleClass1> (argv[3]);
    output2 = new CsaConnectable <SampleClass2> (argv[4]);
    return (0);
}
int SampleApplication : : fini (void)  {
    cout << endl << "Finalizing " << endl << endl;
    delete input1;
    delete input2;
    delete output1;
    delete input1;
    return (0);
}
int SampleApplication : : info(char**, unsigned) const  {
    cout << endl << "Returning infos about " << endl;
    return (0);
}
SampleAppliction : : SampleApplication(void)  {}
SampleApplication : : ~SampleApplication(void)  {}
/* Dynamically linked functions used to control configuration */
extern "C" ACE_Service_Object  *_alloc(void);

ACE_Service_Object *_alloc (void)  {
    return (ACE_Service_Object  *)new SampleApplication;
}
```

```
static SVC_Manager          "-d -p 3333"
dynamic SampleApplication
        ACE_Service_Object * ./SampleApplication.so:_alloc()
        "SampleApplication in1_name in2_name out1_name out2_name"
```

【００３９】ＩＶ．ASCII構築ファイル
【００４０】
【外４】

【００４１】図２には、ソフトウェアＩＣを１つ以上のアプリケーションシステムで実施するブロック回路図が示されている。図２には５つのソフトウェアＩＣが示されている。すなわち、ＩＣ１，ＩＣ２，ＩＣ３，ＩＣ４とＩＣ５である。さらにソフトウェアＩＣを使用する２つのアプリケーション、アプリケーション１とアプリケーション２が示されている。アプリケーション１はソフトウェアＩＣ、ＩＣ１，ＩＣ２，ＩＣ３を含み、アプリケーション２はソフトウェアＩＣ、ＩＣ４とＩＣ５を含む。図からわかるように、アプリケーション１とアプリケーション２は相互作用する。アプリケーション１とアプリケーション２とを含む外部プロセスまたは外部システムも同じようにソフトウェアＩＣの入力と出力を介する。

【００４２】図示のようにＩＣ１は２つの入力側Ｃ１１とＣ１２を有する。ＩＣ１はまたＲ１１を介する１つの出力側を有する。入力側Ｃ１１とＣ１２はＩＣ２の２つの出力側、Ｒ２１とＲ２２にそれぞれ接続されている。ＩＣ２の入力側Ｃ２１はＩＣ１の出力側Ｒ１１と接続されている。
【００４３】ＩＣ３は出力側Ｒ３１を有し、この出力側はＩＣ２の入力側Ｃ２２と接続されている。またＩＣ３の入力側Ｃ３１はアプリケーションを含む外部プロセスと接続されており、入力側Ｃ３２はＩＣ４の出力側Ｒ４１と接続され、出力側Ｒ３２はＩＣ５の入力側Ｃ５２および外部システムと接続されている。出力側Ｒ４１に加えて、ＩＣ４は外部システムに接続された入力側Ｃ４１と、ＩＣ５の入力側Ｃ５１に接続された出力側Ｒ４２を

有する。ＩＣ5はまた、アプリケーションを含む外部プロセスまたは外部システムと接続された出力側Ｒ５１を有する。

【００４４】入力側と出力側は、上に述べたようにCsaConnectableとCsaRemoteを介する。さらに、データが種々の入力側と出力側にダイナミックリンキングを介してオートルーティングされる。これにより、コンフィギュレーションを変化することができ、アプリケーションが再コンパイルまたは再リンキングなしで相互作用することができる。

【００４５】付加的に、前述のソフトウェアＩＣ原理はＡＣＥからのパターン（タスク）と結合することができる。これは、強力なソフトウェアビルディングブロックを得るためであり、このソフトウェアビルディングブロックはハードウェアＰＡＬのような特性を有し、ビルディングブロック内での同期特性とビルディングブロックの外側での非同期特性／相互作用にパワーを与える。

【００４６】内部処理率（ハードウェアＰＡＬでのクロック率に対するカウンタ部分）は従って十分に、接続された環境のイベント入／出力率から独立している。同期を達成するために必要なバッファはまた関連する意味構造なしで与えられる。ハードウェアＰＡＬの同期解決と同じように、同期タスクは必要であればソフトウェアＰＡＬに構築することができる。

【００４７】図3は、ハードウェアＰＡＬとソフトウェアＰＡＬとの比較を示す。図示のように、ハードウェアＰＡＬ３１０はハードウェアＩＣと同じように、２つの入力ピンＩ１とＩ２および２つの出力ピンＯ１とＯ２を有する。しかしハードウェアＰＡＬ３１０内にはまたレジスタ／バッファｒｅｇが設けられており、ここに到来するデータおよび出力されるデータストアされる。

【００４８】カウンタ部分ソフトウェアＰＡＬ３１２は、前に述べたソフトウェアＩＣと同じように入力側Ｒ１，Ｒ２および出力側Ｃ１，Ｃ２を有する。しかしタスクＴ１とＴ２が図示されており、これらはハードウェアＰＡＬ３１０のレジスタ／バッファｒｅｇに代わるものである。他の点ではソフトウェアＰＡＬは上述のソフトウェアＩＣと同じである。

【００４９】ソフトウェアＰＡＬは、アクティブオブジェクトサポートによってソフトウェアＩＣに対するフレキシビリティを内部ロジックに与える。到来するイベントはタスク、例えばタスクＴ１によって、内部ロジックによりさらに処理する前にバッファすることができる。これによりイベントはソフトウェアＰＡＬの内部ロジックから分離される。

【００５０】当業者であれば変形および変更は可能であるが、それらも本願技術に含まれるものである。

【図面の簡単な説明】

【図１】ハードウェアＩＣとソフトウェアＩＣの比較を示す概略図である。

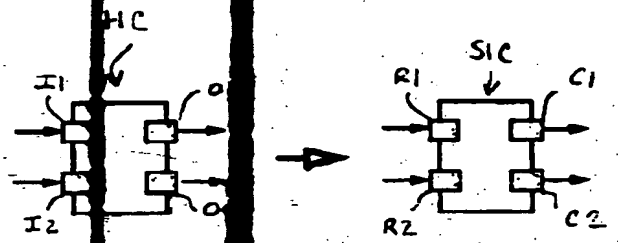
【図２】ソフトウェアＩＣを使用したアプリケーションの概略図である。

【図３】ハードウェアＰＡＬとソフトウェアＰＡＬとの比較を示す概略図である。

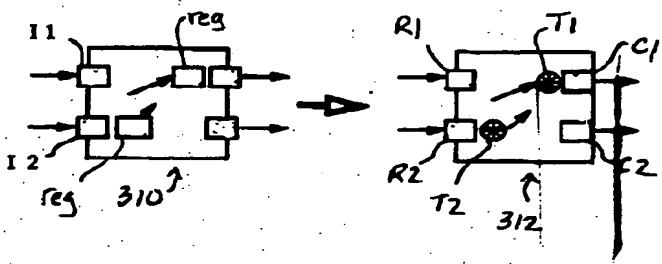
【符号の説明】

ＨＩＣ　ハードウェア集積回路（ＩＣ）
Ｉ１　入力ピン
Ｉ２　入力ピン
Ｏ１　出力ピン
Ｏ２　出力ピン
ＳＩＣ　ソフトウェア集積回路（ＩＣ）
Ｒ１　入力CsaRemote
Ｒ２　入力ＣｓａＲｅｍｏｔｅ
Ｃ１　出力ＣｓａＣｏｎｎｅｃｔａｂｌｅ
Ｃ２　出力CsaConnectable
ＩＣ1～ＩＣ５　ソフトウェアＩＣ
３１０　ハードウェアＰＡＬ
３１２　ソフトウェアＰＡＬ
Ｔ１，Ｔ２　タスク

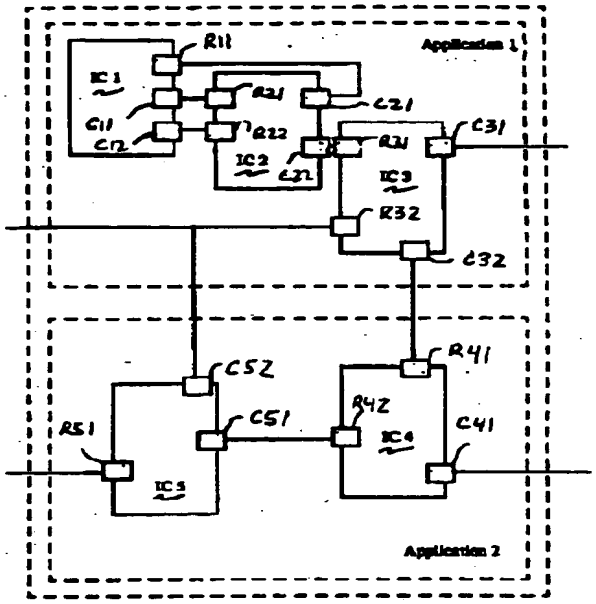
【図１】



【図３】



【図2】



フロントページの続き

(72)発明者　ヨトレ　ベッカー
ドイツ連邦共和国　メーレンドルフ　ヴァッサーフェルクシュトラーセ　10

(72)発明者　ディートリッヒ　クエール
ドイツ連邦共和国　エアランゲン　ニュルンベルガー　シュトラーセ　83

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☑ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.